MIPS TECHNOLOGIES

CODESOURCERY

At the core of the user experience.®

Prelinker Usage for MIPS® Cores
CELF Jamboree #11
Tokyo, Japan, Oct 27, 2006

Arvind Kumar, MIPS Technologies

Kazu Hirata, CodeSourcery

Outline

- What is the prelinker?
- Object formats, loaders & issues
- Driving the prelinker
- Under the hood
- Speedup

Big Picture

- Prelinker is a tool to make programs start faster
- Processes libraries & executable
  - A new step in system building
- Best when whole system is prelinked

New Features

- Supports MIPS
  - Linker, prelinker & dynamic linker changes
- Works in cross environment
  - Original prelinker only ran on the system being prelinked
  - Modified to allow prelinking of a cross built root file system
  - No longer need a live target system

ELF Shared Libraries

- Unspecified load address
  - requires relocations
- Symbolic resolution
  - requires symbol table management

Dynamic Loader

- Allocate space in process's address space
- Load library
- Relocate the Global Offset Table
- Apply relocations from .rel.dyn

a.out shared libraries

- Allocate address at link time
  - Requires universal table of library addresses
- No dynamic relocations
  - No comdat or weak symbols
  - No overriding by executable

Issues

- Symbol resolution takes time
- Relocation causes pages to be unshared
  - GOT is conceptually constant
- Many more shared libraries used now
- Global table of library addresses is unmanageable

Prelinker

- The flexibility of ELF
- The speed of a.out
- Fails gracefully
- Postprocesses linked executables & libraries
  - Preprocesses dynamic loaded images

Modes of Operation

- Prelink all the libraries & executables in these directories
- Prelink this set of libraries to be coresident
- Prelink this executable against these libraries

Dynamic Loader

- Checks consistency of dependent libraries
  - ensures none have changed since executable was prelinked
- If all is well then loads the libraries & executables without needing dynamic linking
- Otherwise falls back to old behaviour
  - Graceful degradation

Limitations

- Cannot automatically prelink dlopen'ed libraries
  - can manually prelink them
- Requires updated linker and dynamic loader
  - Libraries produced by older tools will force full linking

Operation

- Scans libraries and executables
- Builds dependency tree
- Determines which libraries are coresident
- Allocates base addresses
- Prelinks the libraries & executables

Prelinking Libraries

- Adjust every local GOT entry
- Resolve global GOT entries
  - Some entries cannot be prelinked
- Apply relocations according to assigned base addresses
  - Relocations to other libraries
  - Relocations within library
- Insert some ident information

Prelinking Executables

- Apply relocations
  - Relocations to libraries
- Resolve global GOT entries for externaly defined symbols
- Insert ident information
- Create conflict list
  - Executable can override symbols in libraries
  - Global GOT entries that couldn't be prelinked
  - TLS relocations

Ident

- DT_GNU_PRELINKED provides a timestamp
- DT_CHECKSUM provides a checksum
- .gnu.liblist list of libraries needed
- .gnu.libstr names of libraries
- dynamic loader checks
  - each library's DT_GNU_PRELINKED & DT_CHECKSUM against .gnu.liblist

Conversion to RELA

- Sometimes SHT_REL must be converted to SHT_RELA
  - Need correct addend when reprelinking the library or when prelink checks fail
  - Some other (changed) library may override or stop overriding the base symbol
  - So we can't 'undo' the REL relocation to extract the addend
  - We must explicitly store the addend somewhere
  - Or use a relocation type that has no addend

Conflicts

- An executable can override a symbol in a shared library
  - Must update shared library when loaded
  - Prelinker generates such a list for each executable
  - Places it in .gnu.conflicts section
  - Dynamic loader iterates over the list
  - Each conflict causes that page to be duplicated (copy on write)
  - If there are no conflicts, the are no page copies

Conflict Causes

- Improperly linked shared libraries
  - prelinker will warn
- Library (or executable) overriding a symbol
  - eg libpthread.so overrides libc.so
- COMDAT
  - symbol occurs in multiple places, must pick one
- Library's GOT can point to a lazy binding stub
- Thread Local Storage

Conflict Optimizations

- C++ code can have many COMDAT instances
- Some are generated by compiler
  - vtable, rtti info
- These do not have to be commonized
- Prelinker will not emit conflicts for these

Undo

- Prelinker has an undo operation
- .gnu.prelink_undo section contains
  - original ELF, program & section headers
- Can unprelink specific executables & libraries
  - or unprelink everything found in a search path

Size Changes

- Prelinking increases the on-disk image
  - undo info, REL->RELA conversion, conflict list
- A test system of 422 executables and 38 libraries went from 86.8MB to 88.5MB (2%)
  - For libc only, 1749K to 1752K (0.2%)
  - For libstdc++ only, 1413K to 1426K (0.9%)
  - libc avoided the REL->RELA conversion, but libstdc++ did not.

Speedup

- Speedup is noticeable in two places
  - large gui applications with lots of shared libraries
  - shell scripts rapidly invoking many worker tools
- Test machine MIPS64® 20Kc™ core @ 500MHz

Speedup

- Test machine MIPS64® 20Kc™ core @500MHz
- /bin/bash -c /bin/true (300 iterations)
  - 7.4s to 4.6s ~38% improvement
- /bin/bash -c /bin/ls >/dev/null (300 iterations)
  - 10.5s to 7.0s ~33% improvement
- firefox (100 iterations, no X server)
  - 30.9 to 16.7 ~45% improvement

Speedup

- Test machine MIPS32® 24Kc™ core @266MHz
- /bin/bash -c /bin/true (300 iterations)
  - 8.9s to 4.9s ~45% improvement
- /bin/bash -c /bin/ls >/dev/null (300 iterations)
  - 12.3s to 7.2s ~42% improvement
- firefox (100 iterations, no X server)
  - 43.9 to 24.4 ~44% improvement

Where are the Patches?

- Binutils FSF mainline in progress
  - sourceware.org
- GLIBC FSF mainline in progress
  - sourceware.org/glibc
- Prelinker in progress
  - svn://sourceware.org/svn/prelink
- None are yet in official releases